**1. Getting Started Exercises**

**Q1**: What is the difference between a function and a recursion?  
**A**: A function is a block of code that performs a specific task and can be called multiple times. Recursion occurs when a function calls itself to solve a problem by breaking it into smaller sub-problems.

**Q2**: What is the time complexity of finding the last digit of a number?  
**A**: O(1)O(1), since it involves a single arithmetic operation (modulo 10).

**Q3**: What is the significance of the Padovan sequence in mathematics?  
**A**: The Padovan sequence finds applications in geometry, such as in describing the lengths of edges of a stepped series of rectangles.

**2. Searching**

**Q1**: What is Uniform Binary Search?  
**A**: Uniform Binary Search precomputes the midpoints for search operations and uses a lookup table, reducing runtime overhead during repetitive searches.

**Q2**: When is Interpolation Search preferable over Binary Search?  
**A**: When data is uniformly distributed, Interpolation Search can be faster, with an average time complexity of O(log⁡log⁡n)O(\log \log n).

**Q3**: Why does Fibonacci Search use Fibonacci numbers?  
**A**: Fibonacci numbers divide the array into smaller sections that mimic the golden ratio, ensuring optimal partitioning during searches.

**3. Sorting**

**Q1**: Explain the difference between Bubble Sort and Selection Sort.  
**A**: Bubble Sort repeatedly swaps adjacent elements if they are in the wrong order, while Selection Sort selects the smallest (or largest) element and places it in its correct position.

**Q2**: What are the limitations of Insertion Sort?  
**A**: Insertion Sort is inefficient for large datasets due to its O(n2)O(n^2) time complexity in the average and worst cases.

**Q3**: Why is Radix Sort faster than comparison-based algorithms?  
**A**: Radix Sort groups numbers by their digits and sorts them incrementally, achieving O(nk)O(nk) time complexity for nn numbers with kk digits.

**4. Divide and Conquer**

**Q1**: Why does Quick Sort have a worst-case complexity of O(n2)O(n^2)?  
**A**: When the pivot consistently divides the array unevenly (e.g., smallest or largest element chosen as pivot), the recursion depth increases, leading to O(n2)O(n^2) complexity.

**Q2**: How is Merge Sort's space complexity different from Quick Sort's?  
**A**: Merge Sort requires O(n)O(n) auxiliary space for merging, while Quick Sort requires O(log⁡n)O(\log n) for stack space during recursion.

**5. Stack**

**Q1**: What happens if you try to pop an empty stack?  
**A**: This leads to an "underflow" error since there is no element to remove.

**Q2**: Why are stacks used for evaluating postfix expressions?  
**A**: Stacks allow efficient evaluation by storing operands and applying operations in the correct order, as postfix does not require parentheses.

**6. Queue**

**Q1**: What are the advantages of Circular Queues over Linear Queues?  
**A**: Circular Queues utilize memory efficiently by reusing freed-up spaces, whereas Linear Queues can face underutilization.

**Q2**: How does a Deque differ from a regular Queue?  
**A**: A Deque (Doubly Ended Queue) allows insertion and deletion from both ends, unlike a regular queue which follows FIFO.

**7. Linked List**

**Q1**: Why is a singly linked list preferred for stack implementation?  
**A**: A singly linked list allows efficient addition and removal of elements from one end, which aligns with the stack's LIFO behavior.

**Q2**: How do you find the middle node of a linked list efficiently?  
**A**: Use the slow-and-fast pointer method: move one pointer one step and the other two steps. When the faster pointer reaches the end, the slower pointer will be at the middle.

**8. Circular and Doubly Linked Lists**

**Q1**: Why use a Circular Linked List?  
**A**: Circular Linked Lists are useful in applications requiring cyclic traversal, such as round-robin scheduling.

**Q2**: What are the advantages of a Doubly Linked List over a Singly Linked List?  
**A**: A Doubly Linked List allows traversal in both directions and easier deletion/insertion at arbitrary positions, at the cost of extra memory for storing the previous pointer.

**9. Trees**

**Q1**: What is the difference between a Binary Tree and a Binary Search Tree?  
**A**: A Binary Tree is a tree where each node has up to two children. A Binary Search Tree is a binary tree with the additional constraint that the left child is smaller, and the right child is larger.

**Q2**: How do you calculate the depth of a node in a tree?  
**A**: The depth of a node is the number of edges from the root to the node.

**10. Binary Search Trees (BST)**

**Q1**: What is the significance of a BST?  
**A**: A BST allows for efficient searching, insertion, and deletion with O(log⁡n)O(\log n) average time complexity.

**Q2**: What is the difference between deleting a leaf node and a node with two children in a BST?  
**A**: Deleting a leaf node is straightforward, but for a node with two children, the in-order predecessor or successor is used to replace the node.

**11. AVL Trees**

**Q1**: What makes an AVL tree different from a regular BST?  
**A**: An AVL tree is a self-balancing BST where the difference in heights of the left and right subtrees of any node is at most 1.

**Q2**: Why do AVL trees require rotations?  
**A**: Rotations restore balance when the height difference of sub-trees violates the AVL property.

**12. Graphs**

**Q1**: What are the applications of BFS and DFS?  
**A**: BFS is used for shortest path in un-weighted graphs and network broadcasting, while DFS is used for cycle detection and topological sorting.

**Q2**: What is the time complexity of Kruskal’s algorithm for MST?  
**A**: O(Elog⁡E)O(E \log E), where EE is the number of edges in the graph.

**13. Hashing**

**Q1**: What is a hash function?  
**A**: A hash function maps input data to a fixed-size integer value (hash code), which serves as an index for data retrieval.

**Q2**: How does chaining resolve collisions?  
**A**: In chaining, each index of the hash table points to a linked list, where colliding elements are stored.

**14. General Questions**

**Q1**: What factors influence the choice of a data structure for a problem?  
**A**: Factors include time complexity, space complexity, the frequency of operations, and the nature of the problem.

**Q2**: What is Big-O notation?  
**A**: Big-O notation represents the upper bound of an algorithm's time or space complexity, describing its worst-case behavior.